Vamos a realizar una serie de pasos para comenzar con los fundamentos de Java

1. Cuenta en Github

<https://github.com>

1. Mi cuenta (Paco) se llama **serraguti**
2. Necesitamos instalar Git dentro de nuestro equipo

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Debemos indicar nuestro usuario de GIT y nuestro CORREO

Entramos en Terminal (CMD) y escribimos estos comandos

**git config --global user.name USERGIT**

**git config --global user.email** [**MAILGIT@GMAIL.COM**](mailto:MAILGIT@GMAIL.COM)



El siguiente paso es utilizar el entorno de trabajo para Java, que será Visual Studio Code

Descargamos Visual Studio Code

Visual Studio Code trabaja mediante múltiples lenguajes.

Mediante extensiones, podemos trabajar con cualquier lenguaje dentro VS Code

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos configurado Java, simplemente vamos a sincronizar nuestro proyecto individual con Github

Necesitamos instalar otra extensión para trabajar con Github dentro de Vs Code

Esta extensión se validará con nuestra cuenta de Github y podrá sincronizar nuestros proyectos.

Texto

El contenido generado por IA puede ser incorrecto.

Automáticamente nos pedirá validarnos con nuestra cuenta de Git.

Una ventaja radica en que se sincronizarán nuestras extensiones (si queremos) con el resto de equipos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Dentro de Git tenemos dos opciones:

1. **Push**: Nos permite subir nuestros ficheros a Github.com
2. **Pull**:Nos permite descargar nuestros ficheros desde Github.com

Dentro de este proceso, debemos sincronizar los códigos para no tener problemas.

Vamos a subir nuestro nuevo código a un Repositorio llamado **borrarahora**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez publicado, cada vez que realicemos una modificación, siempre tendremos que hacer dos pasos:

1. Escribir un **Commit**. Un commit es una descripción de un estado de una actualización de un Branch/rama dentro de Git
2. Debemos pulsar en **Commit and Push**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Esto es respecto a realizar un Push y trabajar con un equipo (Aula)

El siguiente paso es realizar un Pull (Casa)

En casa no tenemos la carpeta **Prueba**, cerramos la carpeta en VS Code, cerramos VS Code y borramos la carpeta Prueba

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Trucos:

1. Si queremos crear otro repositorio con nuestra carpeta asociada de Github. Razones: Se ha roto, me he liado…

Ahora mismo tenemos asociado **borrarahora**, pero vamos a poner los cambios en otro repositorio distinto y nuevo llamado **borrarluego**

Necesitamos desligar nuestra carpeta de Git. Para ello, debemos eliminar la carpeta oculta **.git** del directorio del proyecto

Command + SHIFT + .

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Cada vez que cambiemos algo entre CASA y el AULA, siempre debemos hacer primero **PULL**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Eliminar los repositorios que tenemos

También podríamos cambiar su estado de visibilidad de private a public o viceversa

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Vamos a trabajar en una carpeta llamada **fundamentosjavagetafe** y abrimos dicha carpeta en VS Code

Dentro de dicha carpeta, separaremos los conceptos mediante nuestras clases de ejemplo.

Utilizaremos una sintaxis progresiva para saber el elemento en el que estamos, por ejemplo: **Class01Descripcion.java**

Si vamos a visualizar tipos de datos primitivos: **Class02TiposPrimitivos.java**

Necesitamos un para de conceptos para empezar:

* El nombre de las clases en Java debe coincidir con el nombre del fichero.
* El nombre de las clases siempre irá en mayúscula cada palabra de la clase.
* Existen elementos llamados **snippets** dentro de VS Code que nos facilitan el trabajo, por ejemplo
  + **sout**: System.out.println
  + **psvm**: public static void main

Comenzaremos con tipos primitivos:

Son clases llamada **Wrapper**, es decir, clases que para ser construidas/instanciadas no es necesario poner el constructor de forma explicita.

Simplemente, con asignar un valor, el objeto es construido.

Una variable en código nos permite almacenar valores para utilizarlos dentro del código.

Dependiendo del valor que deseemos almacenar en la variable, tendremos que declarar dicha variable con un tipado.

Lo más importante no son los tipos, sino la conversión entre los tipos.

Por ejemplo, convertir un texto a un número

Convertir una fecha a un texto

Java tiene dos elementos:

1. Tipo primitivo (pequeño): Solo almacena datos
2. Clase primitiva (grande): Almacena datos y tiene más cositas

Dependiendo del tipo de dato, se tienen una serie de características para dicho tipo de dato.

Por ejemplo, quiero almacenar el número 14:

**int numero = 14;**

GRANDE:

Integer numero = 14;

Integer.MAXVALUE: Nos indica el máximo valor para almacenar

Cada tipo de dato tiene una precisión, no puedo superar dicha precisión

byte peke = 290;

Cuando hablamos de conversión de datos, para convertir entre tipos de dato, se utilizan los **GRANDES**

Los grandes sirven para convertir entre primitivos.

Para visualizar los diferentes tipos de conversión, creamos una clase llamada **Class01Primitivos.java**

Ejemplos de tipos primitivos/Wrapper:

* byte
* short
* int
* long
* double
* decimal
* char
* String
* boolean

**CONVERSION IMPLICITA**

La conversión es automática, no tenemos que hacer nada de nada

Se realiza cuando almacenamos un tipo mayor en un tipo menor (no su valor)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**CONVERSION ENTRE TIPOS PRIMITIVOS COMPATIBLES**

Este tipo de conversión implica convertir un tipo en otro tipo.

Se utiliza lo que se llama **Casting** y permite convertir entre tipos.

(Tipo)valor;

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Tenemos una clase que es como un Primitivo, es decir, podemos crear el objeto sin necesidad de llamar a su constructor, simplemente igualando.

Si yo quiero almacenar textos:

**S**tring texto = “Hola mundo”;

**CONVERTIR CUALQUIER OBJETO A STRING**

Para convertir un objeto a string se utiliza el método **toString()** que tienen TODOS LOS OBJETOS DE Java

El método **toString()** vienen en la clase GRANDE del primitivo.

Texto

El contenido generado por IA puede ser incorrecto.

**CONVERTIR CUALQUIER STRING A PRIMITIVO**

Para convertir cualquier String a Primitivo se utiliza un método de la clase String llamado **valueOf()** que nos permite convertir elementos a String

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**CONVERTIR UN STRING A PRIMITIVO**

Para convertir un string a primitivo necesitamos la clase Grande del primitivo.

Todas las clases primitivas tienen un método llamado **parse()** que nos permite convertir cualquier tipo de dato String a su primitivo.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Dentro de nuestros códigos tenemos tres tipos de errores:

1. Errores de compilación: Son errores de código y que los podemos visualizar de forma más o menos sencilla. (Nos falta un punto y coma)
2. Errores en ejecución: El programa compila, pero se detiene en la ejecución porque tengo algún error. (Problemas al convertir)
3. Errores lógicos: Todo funciona, no nos da ningún error y el programa no hace lo que queremos.

Error de ejecución:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Tenemos objetos especializados como acabamos de visualizar, objetos para almacenar textos, números o decimales.

Cada tipo tiene una funcionalidad, por ejemplo, si declaro un tipo int, puedo sumar, restar o dividir.

Si declaro un tipo String, puedo contar el número de caracteres que contiene.

Tenemos una clase que nos permite “pedir” información al usuario mediante teclado.

Al estar en consola, se utiliza una clase llamada **Scanner** que nos permite pedir información por teclado.

Dentro de Java todo está en librerías. Las librerías están especializadas en sus componentes.

Por ejemplo, existen librerías para ficheros o librerías para acceso a datos o librerías para entornos Web.

Existe una librería llamada **java.util** que nos permite trabajar con herramientas del entorno de Consola

Para poder crear un Scanner necesitamos dos elementos:

1. Utilizar la librería java.util
   1. java.util.Scanner

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* 1. import java.util.Scanner

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

1. Instanciar el objeto Scanner



Para pedir datos al usuario tenemos un método llamado **nextLine()** que nos permite recuperar lo que el usuario ha escrito.

Para concatenar (unir dos textos) se utiliza el símbolo **+** en Java

Texto

El contenido generado por IA puede ser incorrecto.

Vamos a realizar un ejemplo muy sencillo en el que pediremos dos números al usuario por teclado y mostraremos la suma de los dos números.

Creamos un nuevo programa llamado **Class03SumarNumeros.java**

Texto

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class03SumarNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca número 1");**

**//Almacenamos el primer numero**

**//El numero debemos almacenarlo como entero**

**//Voy a capturar el dato como String**

***String* dato = teclado.nextLine();**

**//Convertimos el dato a int**

***int* numero1 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 2");**

**//Volvemos a utilizar la variable dato**

**dato = teclado.nextLine();**

**//Convertimos el dato a int**

***int* numero2 = Integer.parseInt(dato);**

**//Realizamos la suma**

***int* suma = numero1 + numero2;**

**//Dibujamos la suma**

**System.out.println("La suma es " + suma);**

**System.out.println("Fin de programa");**

**}**

**}**

**CONDICIONALES**

Los condicionales en cualquier lenguaje nos permiten que nuestro código sea dinámico y que pueda realizar diferentes acciones dependiendo de lo que vayamos preguntando en las condiciones.

Operadores de comparación

== Igual

> Mayor

>= Mayor o igual

< Menor

<= Menor o igual

!= Distinto

Tenemos varias posibilidades en el momento de preguntar, pero la sintaxis es la misma.

**Sintaxis simple de un Condicional (IF)**

if (condición == true)

{

//Solamente entra si es TRUE

}

**Sintaxis Si no en condición**

if (condición == true)

{

//Acciones para el true

}else {

//Acciones para el false

}

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**Sintaxis con múltiples preguntas**

Se utiliza cuando necesitamos preguntar por más elementos en una condición

if (condiciónA == true)

{

//Acciones para condición A

}else if (condicionB == true) {

//Acciones para condición B

}else {

//Ninguna condición se cumple

}

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a realizar un programa para preguntar si un número es positivo, negativo o cero.

Creamos un nuevo programa llamado **Class04PositivoNegativo.java**

**Texto

El contenido generado por IA puede ser incorrecto.**

**import java.util.Scanner;**

**public class Class04PositivoNegativo {**

**public static *void* main(*String*[] *args*) {**

**//PODEMOS COMBINAR CON SCANNER**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número");**

***String* dato = teclado.nextLine();**

**//Convertimos dato a int**

***int* numero = Integer.parseInt(dato);**

**if (numero > 0){**

**System.out.println("El número es POSITIVO");**

**}else if (numero == 0) {**

**System.out.println("El número es CERO");**

**}else {**

**System.out.println("El número es NEGATIVO");**

**}**

**System.out.println("Fin de programa");**

**}**

**}**

Vamos a realizar un programa en el que pediremos un número del 1 al 4 al usuario.

Debemos indicar la estación dependiendo del número

1.- Primavera

2.- Verano

3.- Invierno

4.- Otoño

Otra opción, escribimos un mensaje diciendo Incorrecto

Llamamos al programa **Class05Estaciones.java**

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class05Estaciones {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número (1-4)");**

***String* dato = teclado.nextLine();**

***int* numero = Integer.parseInt(dato);**

**if (numero == 1){**

**System.out.println("Primavera");**

**}else if(numero == 2){**

**System.out.println("Verano");**

**}else if (numero == 3){**

**System.out.println("Invierno");**

**}else if(numero == 4){**

**System.out.println("Otoño");**

**}else {**

**System.out.println("Opción incorrecta");**

**}**

**}**

**}**

Operadores relacionales

Los operadores relacionales nos permiten preguntar por más de un elemento dentro de una misma pregunta

&& And: Todas las condiciones deben cumplirse

|| Or: Solamente una condición debe cumplirse

! Not: Negación de una condición. Mejor utilizar siempre un Operador

Quiero preguntar si un número es diferente a 1

if (**!** (numero == 1))

if (numero !=1)

Vamos a realizar un programa para calcular el mayor de tres números.

Creamos un nuevo programa llamado **Class06MayorTresNumeros.java**

Tabla

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class06MayorTresNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca número 1");**

***String* dato = teclado.nextLine();**

***int* num1 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 2");**

**dato = teclado.nextLine();**

***int* num2 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 3");**

**dato = teclado.nextLine();**

***int* num3 = Integer.parseInt(dato);**

***int* mayor = 0;**

**//PREGUNTAMOS POR LAS CONDICIONES**

**//22 , 88, 244**

**if (num1 > num2 && num1 > num3){**

**mayor = num1;**

**}else if (num2 > num1 && num2 > num3){**

**mayor = num2;**

**}else {**

**mayor = num3;**

**}**

**System.out.println("El número mayor es " + mayor);**

**}**

**}**